ELEC 8900-7
Physical Design Automation for VLSI and FPGAs

Lecture 3: Partitioning

Mohammed Khalid

Department of Electrical and Computer Engineering
University of Windsor

References and Copyright

- Slide sources (including notes):
  - Prof. Kia Bazargan, University of Minnesota
  - Prof. Rajesh Gupta, University of California, Irvine
  - Dr. Naveed Sherwani (Companion slides with textbook)
  - Prof. Scott Hauck, University of Washington
  - Prof. Majid Sarrafzadeh, UCLA
  - Prof. Steve Kang, Univ. of Illinois (Urbana)
  - Prof. Kurt Keutzer, UC-Berkeley

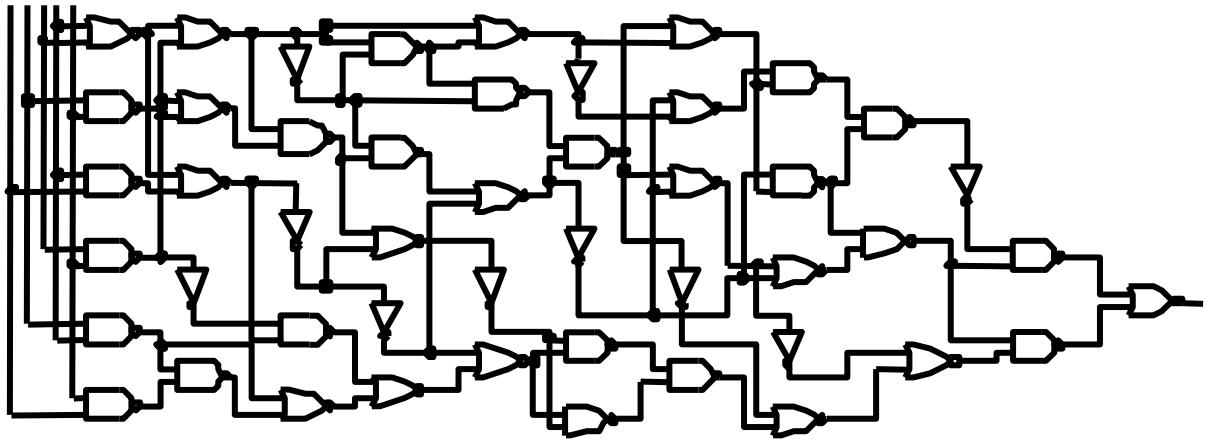
Partitioning

- Decomposition of a complex system into smaller subsystems
  - Done hierarchically
  - Partitioning done until each subsystem has manageable size
  - Each subsystem can be designed independently
- Interconnections between partitions minimized
  - Eases task of interfacing the subsystems
  - Communication between subsystems usually costly

[©Sherwani]

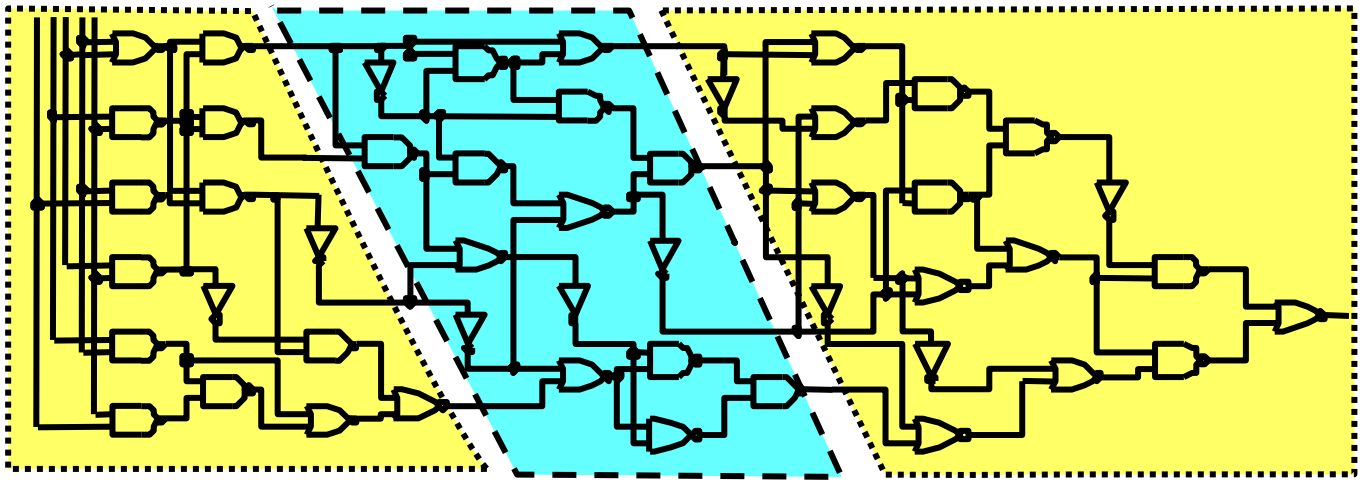
Example: Partitioning of a Circuit

Input size: 48 gates



Cut 1=4 Cut 2=4

Size 1=15 Size 2=16 Size 3=17

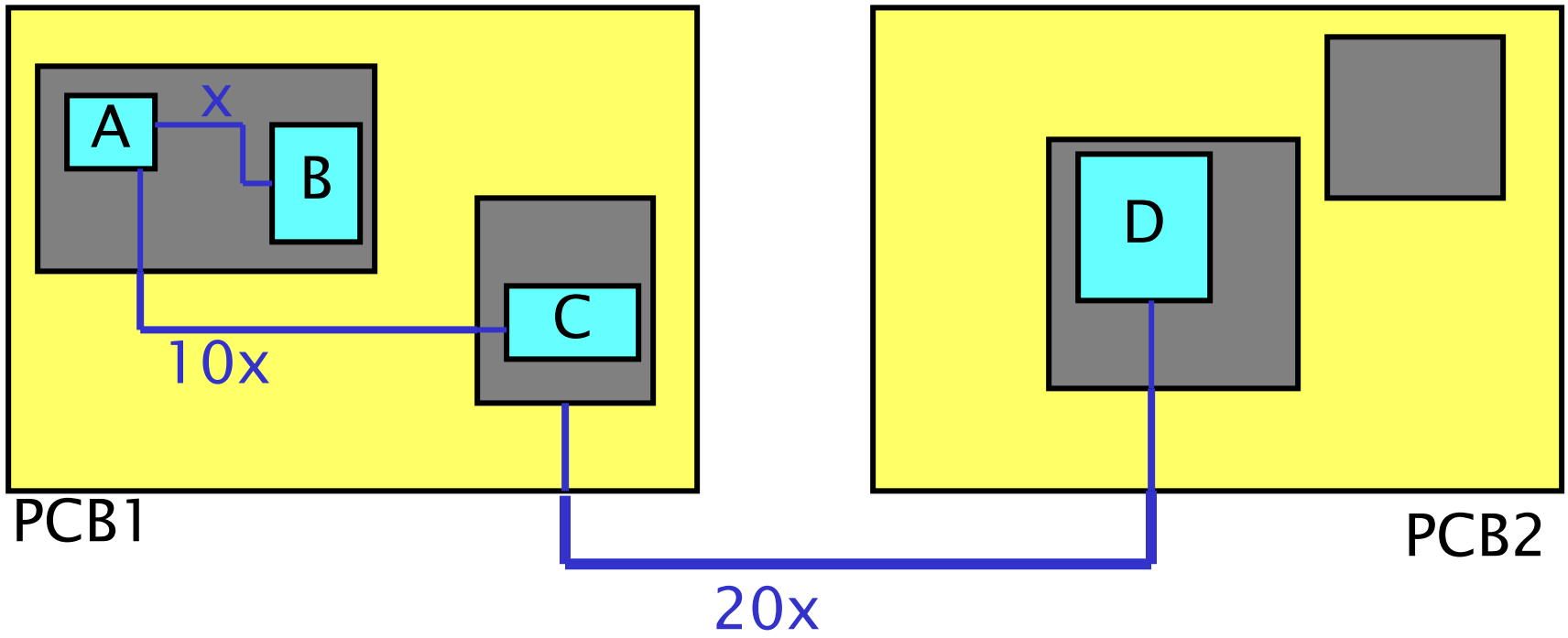


[©Sherwani]

Hierarchical Partitioning

- Levels of partitioning:
  - System-level partitioning:
    Each sub-system can be designed as a single PCB
  - Board-level partitioning:
    Circuit assigned to a PCB is partitioned into sub-circuits each fabricated as a VLSI chip
  - Chip-level partitioning:
    Circuit assigned to the chip is divided into manageable sub-circuits
    NOTE: physically not necessary, but may be done to reduce complexity of layout tasks

Delay at Different Levels of Partitions



- Intra-chip delay – x
- Inter-chip delay – 10x
- Inter-board delay – 20x

[©Sherwani]

Partitioning: Formal Definition

- Input:
  - Graph [G = (V, E)] or hypergraph [H = (V, HE)]
  - Usually with vertex weights
  - Usually weighted edges
- Constraints
  - Number of partitions (2-way/bipartition or K-way)
  - Maximum capacity of each partition
  - Other constraints possible in real world partitioning tools
- Objective
  - Assign nodes to partitions subject to constraints such that the cutsize is minimized
- Tractability
  - Is NP-hard ☹ => use heuristic algorithms

Partitioning Algorithms

- Constructive algorithms
  - Start with a seed node based on connectivity, keep adding nodes based on certain cost functions - give poor results in general, may be used to produce initial partitions for other methods
- Iterative algorithms
  - Start with an initial partition and then successively improve cut size by moving nodes between partitions.
  - Iterative algorithms can be further classified as:
    - **Deterministic** - progresses towards the solution by making deterministic moves, i.e. exact results can be reproduced by re-running the algorithm with same input. Examples: KL, FM
    - **Probabilistic/Stochastic** - progresses towards the solution by making random (coin tossing) moves, i.e. exact results **may not** be reproduced by re-running the algorithm with same input. Examples: Simulated Annealing (SA), Simulated Evolution (SE)

Kernighan-Lin (KL) Algorithm

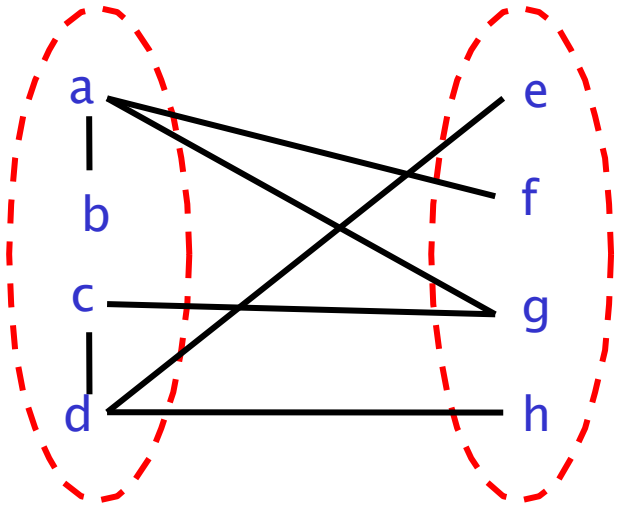
- For non-weighted graphs
- An iterative improvement technique
- A two-way (bisection) partitioning algorithm
- The partitions must be balanced (of equal size)
- KL can be modified to overcome most limitations mentioned above
- Original KL paper (listed below) available on course web site (file: kl_paper.pdf)
  - Read the whole paper in detail - a classic in PDA research literature

B. W. Kernighan and S. Lin, An efficient heuristic procedure for partitioning graphs, Bell System Technical Journal, Vol. 49, No. 2, pp. 291-307, February 1970.

Kernighan-Lin (KL) Algorithm

Procedure: KL heuristic (G);
begin-1 // initial partition obtained randomly or by constructive techniques
bipartition G into two groups V1 and V2, with |V1| = |V2| ± 1;
repeat-2 // main pass
for i = 1 to n/2 do
begin-3
find a pair of unlocked vertices $v_{ai} \in V_1$ and $v_{bi} \in V_2$ whose exchange gives largest decrease in cut-cost or smallest increase;
mark v_{ai} and v_{bi} as locked and store the gain g_i ;
update D-values of all remaining unlocked vertices; // tentative exchange
end-3
find k such that is $\sum_{i=1..k} g_i = Gain_k$ maximized;
if $Gain_k > 0$ **then** // actual exchange
move $v_{a1},...,v_{ak}$ from V_1 to V_2 and $v_{b1},...,v_{bk}$ from V_2 to V_1
until-2 $Gain_k$ is less than or equal to zero
end-1
// key point: accepting negative gains occasionally enables algorithm
// to get out of local (gain) maxima (a.k.a. hill climbing). Eventually
// it may lead to global maximum (gain)

Kernighan-Lin (KL) Example



Final result:
Exchange {d, g} & {c, f},
cut size (cut-cost) goes down from 5 to 1

Step No.	Vertex Pair	Gain	Cut-cost
0	--	0	5
1	{ d, g }	3	2
2	{ c, f }	1	1
3	{ b, h }	-2	3
4	{ a, e }	-2	5

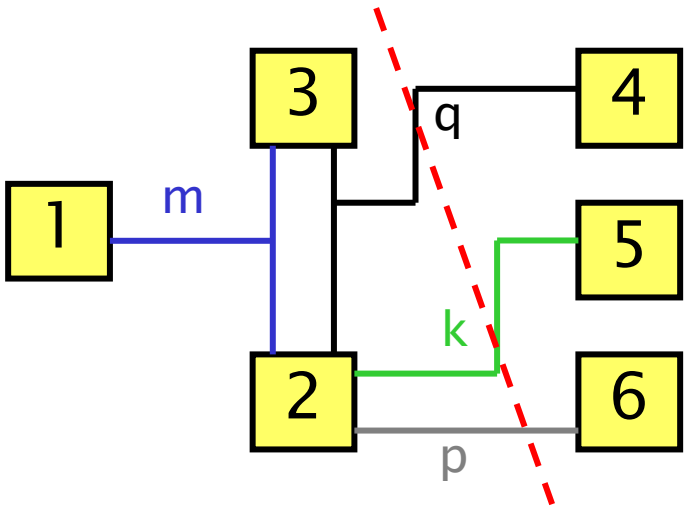
[©Sarrafzadeh]

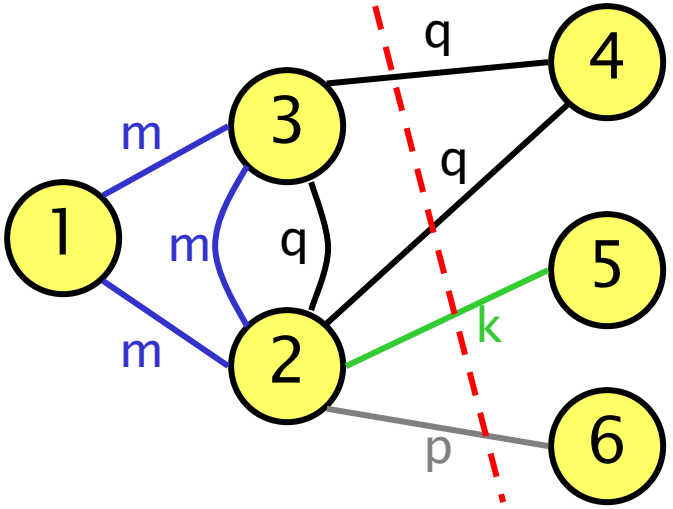
Kernighan-Lin (KL) : Analysis

- Time complexity?
  - For loop is executed $O(n)$ times
    - each iteration takes $O(n^2)$ time (begin-3 to end-3)
  - p passes, no more than few tens (atmost)
  - Total running time $O(pn^3)$
  - Hence time complexity is $O(n^3)$
  - Possible to reduce to $O(n^2 \log n)$
- Drawbacks, most of which can be worked around
  - Balanced partitions only
  - No weight for the vertices
  - High time complexity
  - Only on edges, not hyper-edges

Why Hyperedges?

- For multi terminal nets, K-L may decompose them into many 2-terminal nets, but not efficient!
- Consider this example:
- If A = {1, 2, 3} B = {4, 5, 6}, graph model (right) shows the cutsize = 4 (count q twice) but in the real circuit (left), only 3 wires cut
- Reducing the number of nets cut is more realistic than reducing the number of edges cut

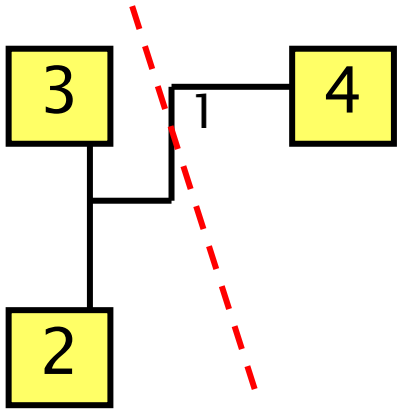




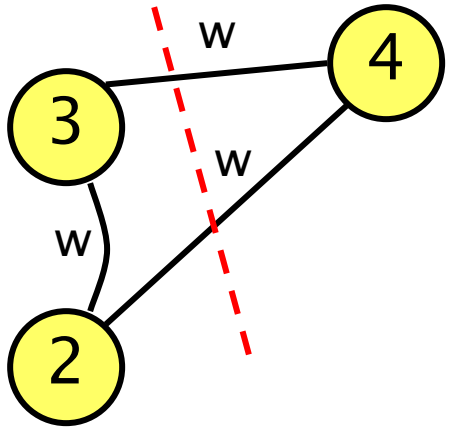
[©Kang]

Hyperedge to Edge Conversion

- A hyperedge can be converted to a "clique" (not recommended)



"Real" cut=1



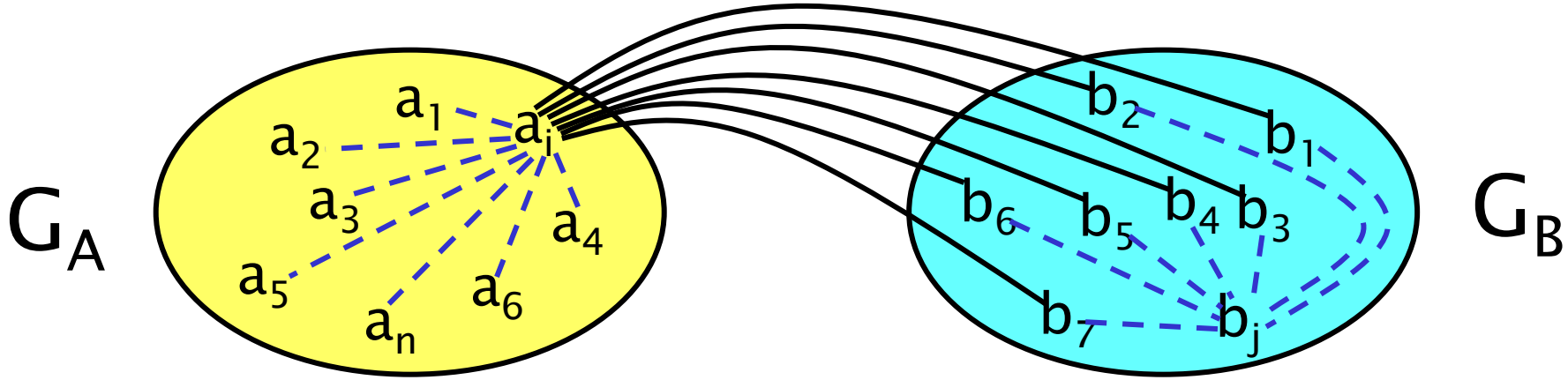
"net" cut=2w

- Better to use approximations when calculating number of connections between two vertices which are connected by multi-terminal nets
- Example given later

Definitions & Notations

- Let G_A and G_B be be two partitions
- Let a_i be a node in G_A and b_j in G_B
- I_{ai} represents the number of internal connections of a_i
- E_{ai} represents the number of external connections of a_i
- **D-value** for a_i, $D_{ai} = E_{ai} - I_{ai}$
- g represents the gain after a_i and b_j are interchanged, i.e. the actual reduction in cut size (cut-cost)
- The following terms are equivalent
  - node & vertex
  - swap & exchange
  - cut size & cut-cost

Gain Calculation for a swap



$$I_{a_i} = \sum_{x \in A} C_{a_i x}, \qquad E_{a_i} = \sum_{y \in B} C_{a_i y}$$

Likewise,

$$D_{a_i} = E_{a_i} - I_{a_i}$$

$$D_{b_j} = E_{b_j} - I_{b_j} = \sum_{x \in A} C_{b_j x} - \sum_{y \in B} C_{b_j y}$$

[©Kang]

Gain Calculation for a swap

- Lemma: Consider any $a_i \in A$, $b_j \in B$. If a_i, b_j are interchanged, the gain is

$$g = D_{a_i} + D_{b_j} - 2C_{a_i b_j}$$

- Proof:

Total cut cost before interchange (T) between A and B

$$T = E_{a_i} + E_{b_j} - C_{a_i b_j} + (\text{cost for all others})$$

Total cut cost after interchange (T') between A and B

$$T' = I_{a_i} + I_{b_j} + C_{a_i b_j} + (\text{cost for all others})$$

Therefore

$$g = T - T' = \underbrace{E_{a_i} - I_{a_i}}_{D_{a_i}} + \underbrace{E_{b_j} - I_{b_j}}_{D_{b_j}} - 2C_{a_i b_j}$$

[©Kang]

D-value updates after node swap

- Lemma:
  - Let D_x', D_y' be the new D values for elements of A - $\{a_i\}$ and B - $\{b_j\}$. Then after interchanging a_i & b_j,

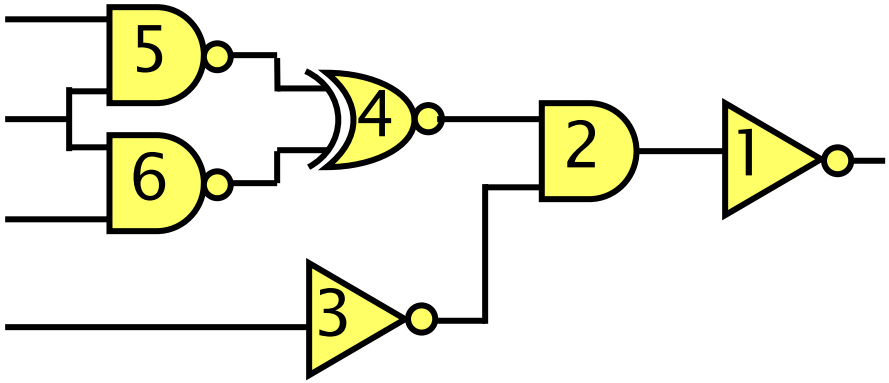
$$D_x' = D_x + 2C_{xa_i} - 2C_{xb_j} \quad , \quad x \in A - \{a_i\}$$

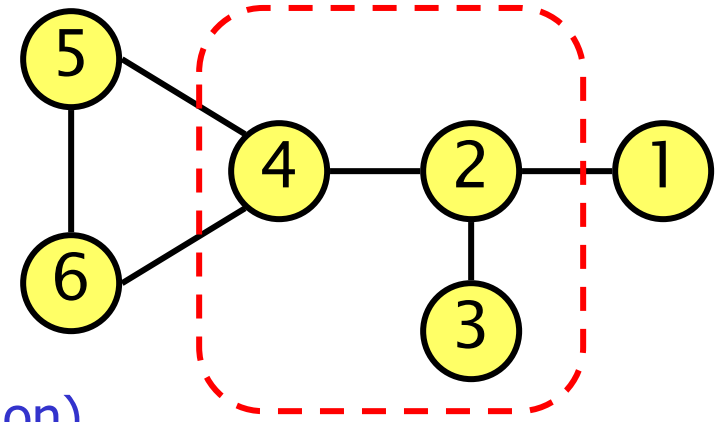
$$D_y' = D_y + 2C_{yb_j} - 2C_{ya_i} \quad , \quad y \in B - \{b_j\}$$

- Proof:
  - see paper

[©Kang]

Example: KL





Initial partition

- Step 1 - Initialization (any random partition)

  A = {2, 3, 4}, B = {1, 5, 6} (cut size = 3)

  A′ = A = {2, 3, 4}, B′ = B = {1, 5, 6}

- Step 2 - Compute D values

  $D_1 = E_1 - I_1 = 1-0 = +1$

  $D_2 = E_2 - I_2 = 1-2 = -1$

  $D_3 = E_3 - I_3 = 0-1 = -1$

  $D_4 = E_4 - I_4 = 2-1 = +1$

  $D_5 = E_5 - I_5 = 1-1 = +0$

  $D_6 = E_6 - I_6 = 1-1 = +0$

[©Kang]

Example: KL (cont.)

- Step 3 - compute gains (for each possible unlocked pair)

$g_{21} = D_2 + D_1 - 2C_{21} = (-1) + (+1) - 2(1) = -2$

$g_{25} = D_2 + D_5 - 2C_{25} = (-1) + (+0) - 2(0) = -1$

$g_{26} = D_2 + D_6 - 2C_{26} = (-1) + (+0) - 2(0) = -1$

$g_{31} = D_3 + D_1 - 2C_{31} = (-1) + (+1) - 2(0) = 0$

$g_{35} = D_3 + D_5 - 2C_{35} = (-1) + (0) - 2(0) = -1$

$g_{36} = D_3 + D_6 - 2C_{36} = (-1) + (0) - 2(0) = -1$

$g_{41} = D_4 + D_1 - 2C_{41} = (+1) + (+1) - 2(0) = +2$

$g_{45} = D_4 + D_5 - 2C_{45} = (+1) + (+0) - 2(+1) = -1$

$g_{46} = D_4 + D_6 - 2C_{46} = (+1) + (+0) - 2(+1) = -1$

- The largest g value is $g_{41} = +2$

$\Rightarrow$**interchange** 4 and 1 & **lock** $(a_1, b_1) = (4, 1)$

$A' = A' - \{4\} = \{2, 3\}$

$B' = B' - \{1\} = \{5, 6\}$ both not empty

[©Kang]

Example: KL (cont.)

- Step 4 - update D values of node connected to vertices (4, 1)

  $D_2' = D_2 + 2C_{24} - 2C_{21} = (-1) + 2(+1) - 2(+1) = -1$
  $D_5' = D_5 + 2C_{51} - 2C_{54} = +0 + 2(0) - 2(+1) = -2$
  $D_6' = D_6 + 2C_{61} - 2C_{64} = +0 + 2(0) - 2(+1) = -2$

- Assign $D_i = D_i'$, repeat step 3 :

  $g25 = D_2 + D_5 - 2C_{25} = -1 - 2 - 2(0) = -3$
  $g26 = D_2 + D_6 - 2C_{26} = -1 - 2 - 2(0) = -3$
  $g35 = D_3 + D_5 - 2C_{35} = -1 - 2 - 2(0) = -3$
  $g36 = D_3 + D_6 - 2C_{36} = -1 - 2 - 2(0) = -3$

- All values are equal;
  arbitrarily choose $g_{36} = -3 \Rightarrow$

  $A' = A' - \{3\} = \{2\}, \quad B' = B' - \{6\} = \{5\}$

  (a2, b2) = (3, 6)

  New D values are:

  $D_2' = D_2 + 2C_{23} - 2C_{26} = -1 + 2(1) - 2(0) = +1$
  $D_5' = D_5 + 2C_{56} - 2C_{53} = -2 + 2(1) - 2(0) = +0$

- New gain with $D_2 \leftarrow D_2'$, $D_5 \leftarrow D_5'$

  $g_{25} = D_2 + D_5 - 2C_{52} = +1 + 0 - 2(0) = +1 \Rightarrow$ (a3, b3) = (2, 5)

[©Kang]

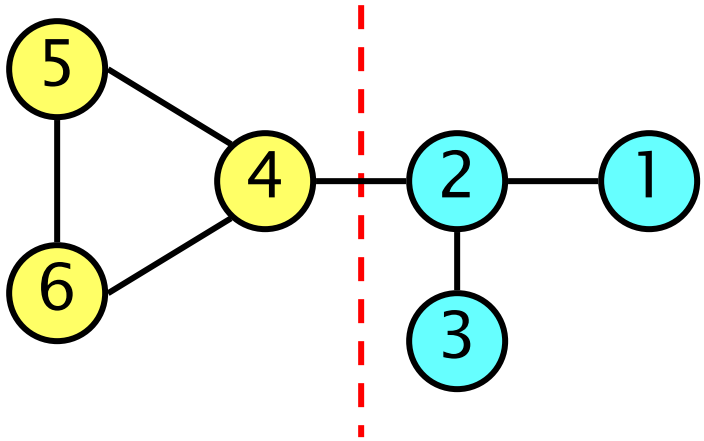
Example: KL (cont.)

- Step 5 - Determine the # of moves to take

  $g_1 = +2$

  $g_1 + g_2 = +2 - 3 = -1$

  $g_1 + g_2 + g_3 = +2 - 3 + 1 = 0$



- The value of k for max G is 1

  $X = \{a_1\} = \{4\}, Y = \{b_1\} = \{1\}$

- Move X to B, Y to A $\Rightarrow$ A = {1, 2, 3}, B = {4, 5, 6}
- Repeat the whole process:
  - • • • • •
- The final solution is A = {1, 2, 3}, B = {4, 5, 6}
  - Final cut size = 1 (intial cut size was 3)
- Toy example, real circuits have thousands of nodes in each partition

Extending KL for more general cases

- Unbalanced partitions - add dummy nodes (nodes with no connections) and swap with dummy nodes if necessary
- Dealing with hyperedges (multi-terminal nets)
  - Use approximations when calculating Ea and Ia
  - Simply means higher the fanout, less important the net

- Multi-way (k-way) partitioning
  - Obtain initial partition randomly or by recursive bipartitioning followed by improvement using pair-wise swapping
  - Much harder than bipartitioning problem (in practice)
  - Other methods (algorithms) also used.

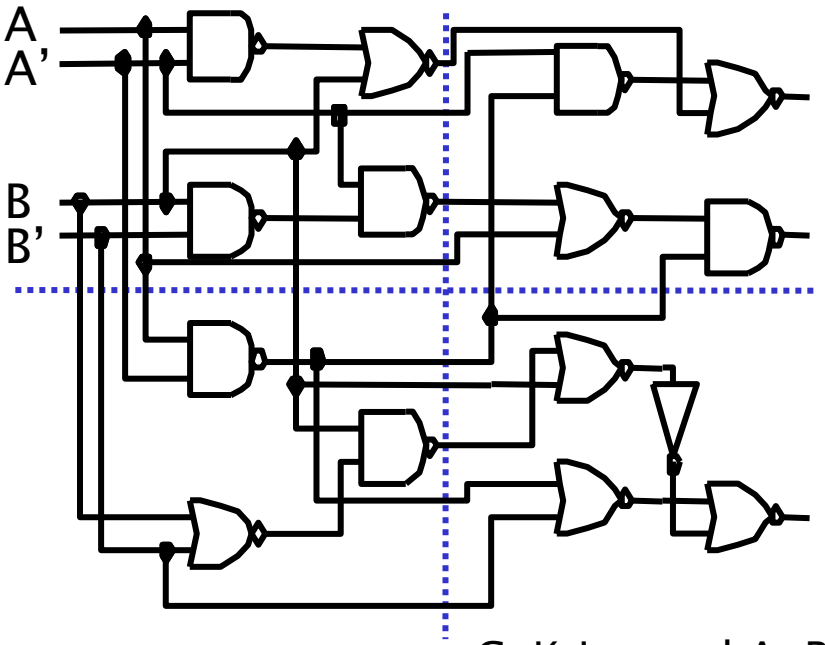
Fiduccia-Mattheyses (FM) Algorithm

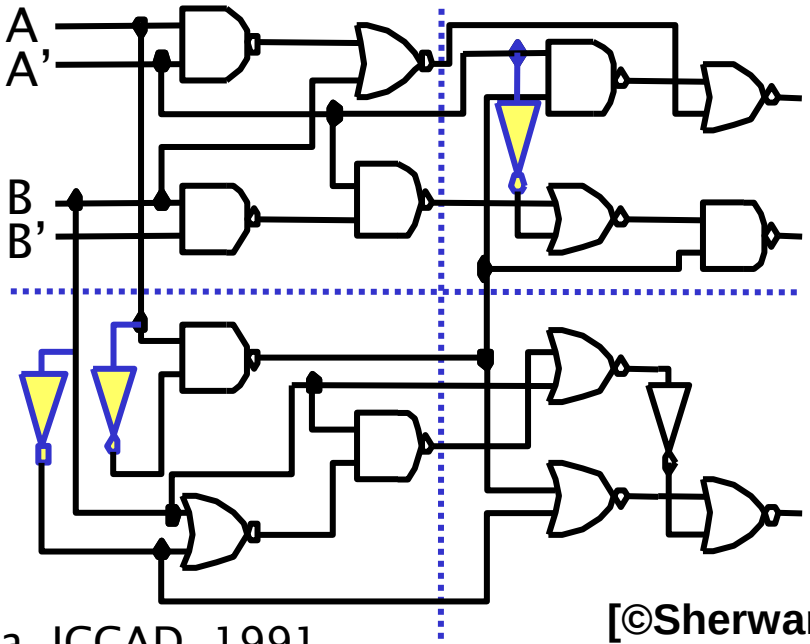
- Modified version of KL
- A single vertex is moved across the cut in a single move
  - ➔ Can handle unbalanced partitions
- Vertices are weighted
- Concept of cutsize extended to hypergraphs
- Special data structure to improve time complexity to O(n), where n is number of terminals NOT nodes
  - Main feature
- Can be extended to multi-way partitioning

C. M. Fiduccia and R. M. Mattheyses, A linear time heuristic for improving network partitions, Proc. of 19th Design Automation Conference (DAC), 1982.

Subgraph Replication to Reduce Cutsize

- Vertices are replicated to improve cutsize
- Good results if limited number of components replicated
- Increases layout area due to extra gates - may be compensated by reduction in interconnect area
  - with newer fab. tech. gates are almost free, wires dominate



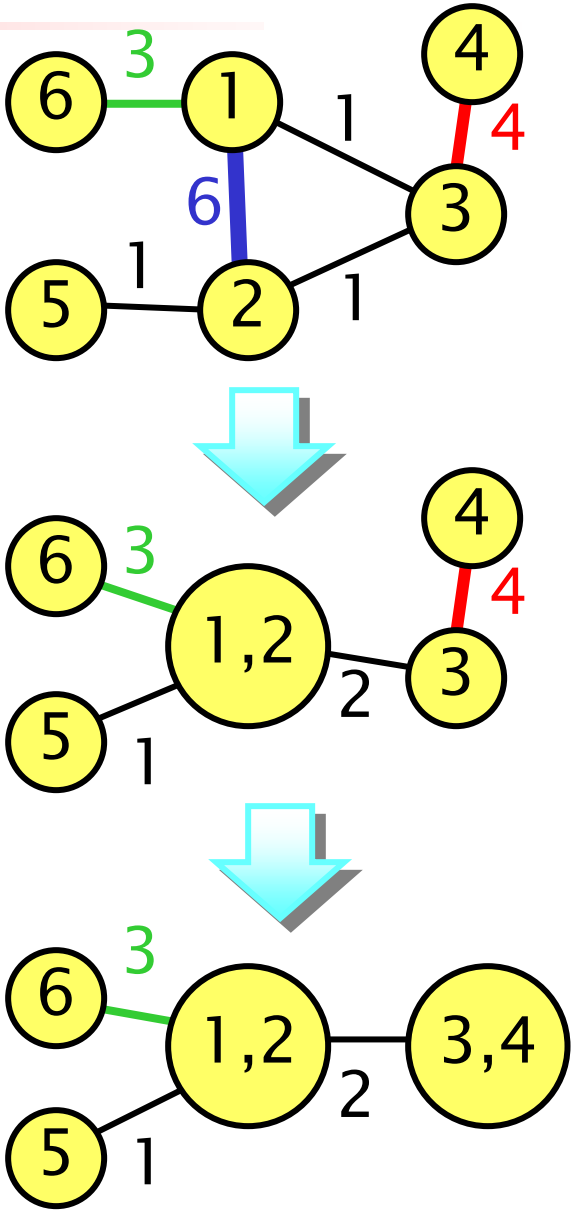


C. Kring and A. R. Newta, ICCAD, 1991.

[©Sherwani]

Clustering

- Clustering
  - Bottom-up process
  - Merge heavily connected components into clusters
  - Each cluster will be a new "node"
  - "Hide" internal connections (i.e., connecting nodes within a cluster)
  - "Merge" two edges incident to an external vertex, connecting it to two nodes in a cluster
- Can be a preprocessing step before partitioning
  - Each cluster treated as a single node
  - Usually gives significant quality and speed improvements.



Other Partitioning Methods

- KL and FM have each held up very well - **still** most widely used in real partitioning tools
  - Reasons: Fast & flexible, easy to add real world constraints
- Min-cut / max-flow algorithms
  - Ford-Fulkerson – for unconstrained partitions
- Ratio cut
- Genetic algorithm (simulated evolution)
- Simulated annealing
- Challenges (future research opportunities)
  - Effective algorithms for more constrained partitioning problems, e.g. timing, clocking, etc. (other than just cut size) - application or target dependent tuning required in real tools.
  - Effective algorithms for constrained multi-way partitioning, e.g. partitioning into multiple FPGAs

To Probe Further...

- C. J. Alpert and A. B. Kahng, ”Recent Developments in Netlist Partitioning: A Survey", Integration: the VLSI Journal, 19 (1-2), 1995, pp. 1-81. (http://vlsicad.cs.ucla.edu/~cheese/survey.html)

- George Karypis and Vipin Kumar,
"Multilevel k-way hypergraph partitioning",
Design Automation Conference, pp. 343-348, 1999.

- J. Cong and S. Lim, ”Multiway Partitioning with Pairwise Movement",
Proc. of ACM/IEEE ICCAD, pp. 355-359, 1998.

- A. E. Caldwell, A. B. Kahng, I. L. Markov,
"Design and Implementation of Move-Based Heuristics for VLSI Hypergraph Partitioning",
ACM Journal on Experimental Algorithms, Vol. 5, 2000.